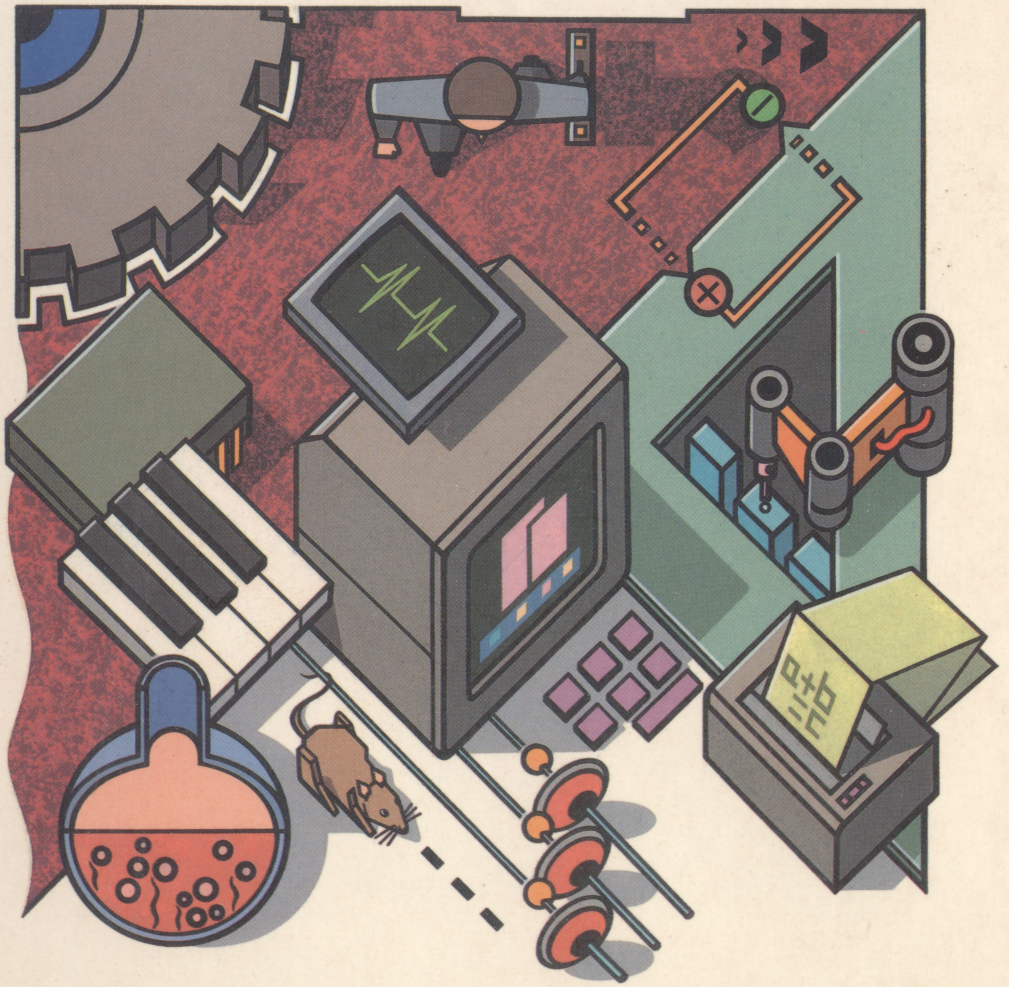
USER PORT/MIDI EXPANSION CARD

USER GUIDE



A3000

Acorn
The choice of experience.

USER PORT/
MIDI EXPANSION CARD

USER GUIDE

Copyright © Acorn Computers Limited 1989

Neither the whole nor any part of the information contained in, nor the product described in this Guide may be adapted or reproduced in any material form except with the prior written approval of Acorn Computers Limited.

The products described in this guide are subject to continuous development and improvement. All information of a technical nature and particulars of the products and their use (including the information and particulars in this Guide) are given by Acorn Computers Limited in good faith. However, Acorn Computers Limited cannot accept any liability for any loss or damage arising from the use of any information or particulars in this guide, or any incorrect use of the products. All maintenance and service on the products must be carried out by Acorn Computers' authorised dealers. Acorn Computers Limited can accept no liability whatsoever for any loss or damage caused by service, maintenance or repair by unauthorised personnel.

If you have any comments on this Guide, please complete and return the form at the back of the Guide to the address given there. Any other correspondence should be addressed to:

Customer Support and Service
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN

Information can also be obtained from the Acorn Support Information Database (SID). This is a direct dial viewdata system available to registered SID users. Initially, access SID on Cambridge (0223) 243642: this will allow you to inspect the system and use a response frame for registration.

ACORN and ECONET are trademarks of Acorn Computers Limited.

Within this publication, the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

Published by Acorn Computers Technical Publications Department
First published 1989
Part number 0480,351
Issue 1

Contents

Introduction	Introduction to the expansion card	1
	Installation	1
	About this Guide	1
Introducing MIDI	What is MIDI?	3
	What the MIDI interface can do	3
	Connecting up	3
	Making sounds	4
Programmer's guide to MIDI	Writing programs	7
	Application software	7
	Software structure	7
	Data format	8
	Data types	8
	Channel modes	9
	MIDI interpreter and sound system driver	9
	MIDI ports	10
	MIDI *commands	11
Software interrupts (SWIs)	General	13
	General interface commands	14
	Data reception commands	28
	Data transmission commands	31
	Service calls	54
	Events	55
	Summary of SWI commands	56
	Programming with software interrupts	57
Timing	Timing	59
	Reception of special messages	62
MIDI interface data specification	Summary specification	65
	Summary of status bytes	65

User Port

General 67
The A3000 implementation 67
Incompatibilities with the BBC User Port 68
Using the interface 68
An example 70
Technical specification 71
User Port address allocation 71

Appendix A

Block diagram 73

Bibliography

MIDI 75
User port 75

Introduction

Introduction to the expansion card

The User Port / MIDI expansion card fits inside your A3000 computer and expands its capability by providing:

- An 8-bit User Port, compatible with the user port on the Archimedes I/O expansion card, and largely compatible with the User Port interface on the BBC Model B and Master 128 microcomputers, enabling you to connect your A3000 computer to a wide range of peripheral equipment already available for these computers.
- MIDI (Musical Instrument Digital Interface), with IN, OUT and THRU connections, compatible with the International MIDI Association specification (and with those fitted to the Archimedes MIDI expansion card). MIDI enables you to connect synthesisers, rhythm machines, home computers and sequencers together through a standard interface.

Installation

The A3000 User Port / MIDI expansion card must be fitted by an Acorn Authorised Dealer. Take your A3000 computer (in its original packaging) to an Acorn Dealer who will install it for you. The Dealer may make a charge for this service.

About this Guide

The User Port / MIDI expansion card User Guide is intended to provide an introduction to MIDI for first-time users, as well as a reference for experienced users and programmers who may wish to write software both for the MIDI interface and the User Port. This Guide should be read in conjunction with the *A3000 User Guide* and the *Programmer's Reference Manual.*

Introducing MIDI

What is MIDI ?

MIDI is the acronym for Musical Instrument Digital Interface. It is a means of connecting electronic instruments, such as synthesisers and drum machines, to each other and to sequencers and computers, so that they can interact. Thus music can be stored and edited by a computer, to be played on a synthesiser or rhythm machine, or a synthesiser can control a rhythm machine via MIDI. This is possible because MIDI provides a standard form of communication.

Each MIDI instrument will have a transmitter and a receiver, or occasionally just one of these, using a standard code to convey information about keys pressed, note lengths and a variety of other possible messages, such as which of the different voices of a synthesiser is to play the note.

These voices are the different programmed sounds which the instrument can make, such as piano or bongo.

What the MIDI interface can do

By installing the MIDI interface in your A3000 computer and connecting a music keyboard, you can use the computer as a musical instrument. The sound can be played through the speakers on the computer, through headphones or your hi-fi system.

The A3000 computer can also be used to extend the capabilities of the instruments you have, provided they have MIDI installed. By writing or buying application programs much more becomes possible. You can write such programs yourself using the high speed BASIC V on the A3000 computer.

Connecting up

Warning: MIDI sockets carry digital information; audio connections (such as those from your hi-fi system) must NEVER be made to these sockets as serious damage could result.

MIDI instruments must be connected to your A3000 computer via MIDI leads. These may look the same as your ordinary hi-fi leads, but the cables are different. The MIDI leads will have a five pin DIN plug on each end, connected by a shielded, twisted-pair cable, and are available from your local music shop.

Your music keyboard or synthesiser must have MIDI installed. This is sometimes an optional extra, so check before you buy.

Connect the MIDI IN socket on the instrument to the MIDI OUT socket on the User Port / MIDI expansion card, and the MIDI OUT socket on the instrument to the MIDI IN socket on the expansion card:

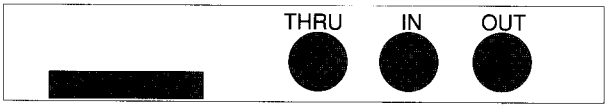


figure 1: Expansion card ports

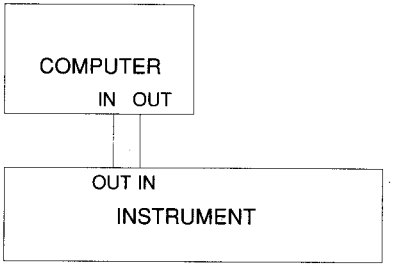


figure 2: Connecting an instrument

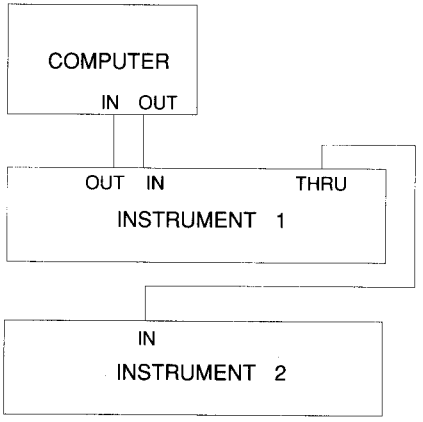


figure 3: Connecting more than one instrument

To connect more than one instrument at a time the instruments should be fitted with MIDI THRU sockets (see figure 3). Refer to the instrument user guide for details. You can use the speakers on the A3000 computer to provide the sound, or you can plug personal stereo headphones into the Headphones 32Ω socket on the back of your A3000 computer, using a 3.5mm jack plug. You can also connect this socket to certain inputs on hi-fi amplifiers. Consult the *A3000 Welcome Guide* for further information.

The outputs of various instruments, including the A3000 computer, can be combined in an audio mixer and used to drive a hi-fi system in the normal way.

Making sounds

To check that you have connected your interface satisfactorily, type:

```
*modules
```

The system will now list the modules you have available, which should include:

MIDI

If not, your MIDI interface has not been installed correctly. Refer back to your supplier.

Assuming that MIDI is on the list, type:

```
*midisound in
```

Now when you play the music keyboard it will play the A3000 computer sound system. You can either type:

```
*speaker on
```

to hear it from the built-in speakers or, for higher quality stereo sound, plug a pair of headphones or your hi-fi system into the Headphones 32Ω socket at the back of the computer.

If, when you type `*voices`, a number of voices is listed on your screen, then you can switch your system between these voices using the program change buttons on your instrument.

Programmer's guide to MIDI

Writing programs

The very fast BASIC V available on the A3000 computer enables you to write application programs in BASIC which use your MIDI interface. To do this, use the software interrupts listed in the next chapter.

Before writing any major application programs, we recommend that you obtain a copy of the MIDI specification from the address given in the Bibliography at the end of the Guide. A useful book on MIDI is mentioned there as well.

Application software

The full potential of the MIDI system can be realised using application software. Only packages specifically written for the A3000 computer RISC OS operating system are suitable.

This chapter should be read in conjunction with the *Programmer's Reference Manual*.

Software structure

The RISC OS MIDI interface consists of hardware and software to give an interface for both input and output of data to MIDI specification 1.0.

The software is in six main parts:

- A low-level interrupt routine which buffers data to and from the UART (Universal Asynchronous Receiver Transmitter), which transmits and receives the raw MIDI data. This routine also deals invisibly with transmission and reception of the System Real Time messages used to synchronise the system.
- A combined MIDI interpreter and sound system driver which can be driven from the incoming or outgoing MIDI data.
- A millisecond routine which time-stamps the incoming MIDI data, and which can time scheduled outgoing data.
- A sound interrupt (SIRQ) synchronous routine for background operation, and which can time scheduled outgoing data.
- A set of SWI (SoftWare Interrupt) calls (listed in the next chapter) to provide an interface between the application program and the MIDI interface.

- A set of operating system *commands.

The MIDI software is controlled and communicated with using SWI calls, and can also be controlled, to a limited extent, with *commands and via the MIDI interface. The software provides low-level support for receiving and transmitting data bytes allowing great flexibility for an application program.

Data format

MIDI messages have one status byte followed by one or two data bytes, except System Real Time and System Exclusive messages – see below.

- Channel Voice messages contain a four-bit number in the status byte which addresses the message to one of 16 channels. An instrument is assigned to a basic channel over which mode messages are sent. If the instrument has several voices these can be assigned extra channels to control the voices separately.
- System messages are not given channel numbers; there are three types, as follows:
  - Common messages are for all the units in a system.
  - Real Time messages are for all the units in a system and contain only status bytes. They can be sent at any time, even between bytes of another message.
  - Exclusive messages include a manufacturer's identification code, and any number of data bytes, terminated by an 'end of exclusive', or another status byte. Only equipment which recognises the identification will accept the data, the format of which is specified by the manufacturer.

Data types

Status bytes

Status bytes are eight bits with the most significant bit (MSB) set (1). These identify the message type. Status bytes command the receiver to adopt their status, except for Real Time messages. Unimplemented or undefined status bytes are ignored.

Running status

For voice and mode messages, the receiver will remain in the status of the last status byte until a different one is received. Therefore when many messages with the same status are being sent, the status byte may be omitted. This is particularly useful for long strings of Note On/Off messages. Real Time messages will only change the running status temporarily.

Data bytes

One or two data bytes follow the status bytes, except in Real Time messages. Data bytes are eight bits, with the most significant bit set to 0. Each status byte must be followed by the correct number of data bytes, and action will wait until they are all received.

Channel modes

Mode messages control the way in which Note On/Off information is routed to the instruments and their voices via the 16 MIDI channels. The mode messages available for this purpose are defined in the table below. A MIDI receiver or transmitter can only operate in one mode at a time.

Each receiver is assigned to a basic channel N, and may have a number of voices M assigned to adjacent channels.

Mode	Omni		
1	On	Poly	Voice messages are received from all channels and assigned to voices polyphonically by the receiver.
2	On	Mono	Voice messages are received from all voice channels and control only one voice, set by the receiver, monophonically.
3	Off	Poly	Voice messages are received in voice channel N only, and are assigned to voices polyphonically by the receiver.
4	Off	Mono	Voice messages are received in voice channels N to N+M–1 and assigned monophonically to voices 1 to M, respectively. The number of voices M is specified by the Mono Mode message, one voice per channel. This means that each voice in the receiver can be controlled separately by the transmitter.

Each transmitter is also assigned to a basic channel N. Those without the capacity to select channels will normally use channel one.

MIDI interpreter and sound system driver

This background process allows MIDI data from the MIDI IN or MIDI OUT sockets to drive the sound system directly, without user intervention, and with the computer still able to perform other tasks normally. This facility is disabled by default, and is enabled from the command line using `*midisound`. The state of this driver does not affect the data received by MIDI_RxCommand and MIDI_RxByte.

The MIDI interpreter does not respond to notes of very short duration (less than about 20ms). This means that certain drum machines cannot be used to trigger notes on the computer sound voices.

MIDI implementation chart

An implementation chart for the built-in MIDI interpreter is shown on a separate sheet accompanying the expansion card, laid out in accordance with the MIDI specification.

Some MIDI commands are ignored by the MIDI interpreter. These are listed below:

System messages:

- Song Position Pointer
- Song Select
- System Exclusive.

Real time messages:

- Timing Clock
- Start
- Continue
- Stop
- System reset.

Channel voice messages:

- Control Change 0-121
- Polyphonic Key Pressure (After touch)
- Channel Pressure (After touch)

and all bytes undefined in MIDI specification 1.0.

These commands are, however, available for use by application software.

Pitch bend

The pitch bend algorithm used by the sound system driver has the shortcoming that if the notes are newly triggered while pitch bend is in operation, they will start with the normal pitch, and will only be pitch changed when the next pitch bend message is received.

MIDI ports

With one User Port / MIDI expansion card installed, you will have one MIDI port, numbered 0. Up to four MIDI ports are supported, to cover the

future possibility of connecting extra expansion cards or custom hardware to provide extra ports. Ports are numbered consecutively, from 0 to 3. Application software supporting multiple ports should use the SWI MIDI_Init to find the number of recognised ports currently installed.

MIDI *commands

MIDI *commands can be divided into two groups:

- Interpreter control
- Timing clock generator control.

These are detailed below:

Interpreter control

*MidiSound

Parameter 1 'in', 'out' or 'off'; to specify incoming data, outgoing data or disable a port.

Parameter 2 (optional) <1–4> to specify MIDI port number. This enables or disables the MIDI interpreter and specifies the source of MIDI data.

*MidiTouch

Parameter is 'on' or 'off'. This enables or disables the touch-sensitivity of the interpreter (responds to Note On velocity).

*MidiChannel

Used to set the Basic Channel of the MIDI interpreter. Parameter is 1–16.

*MidiMode

Used to set the MIDI mode of the MIDI interpreter. Parameter is 1–4.

Timing clock generator control

*MidiStart <time>

Used to transmit a Start message and start automatic transmission of Timing Clock messages every <time> milliseconds. Parameter, if specified, is integer time in milliseconds 1– &FFFF, or 0 to leave unchanged.

*MidiStop

Used to transmit a Stop message and stop automatic transmisssion of MIDI Timing Clock messages.

*MidiContinue

Used to transmit a Continue message and to restart automatic transmission of Timing Clock messages.

Software interrupts (SWIs)

General

Software interrupts provide an interface between an application program and the MIDI interface itself.

Values given for RO and R1 are the contents of those CPU registers; where no values are given the command does not use the registers.

SWI calls used for setting up and controlling the MIDI are detailed on the following pages. They have been divided into the following groups:

- General interface commands
- Data reception commands
- Data transmission commands
- Service calls
- Events.

MIDI_SoundEnable (&404C0)

Enable the MIDI interpreter/sound system driver, so that MIDI data will be interpreted, and control the sound system. The mode will default to 1 (Omni On:Poly), but may be changed under MIDI or SWI control.

On entry

R0 bit 0–1	= 0	to disconnect sound system driver.
	1	for connection to the MIDI IN buffer.
	2	for connection to the MIDI OUT buffer.
R0 bit 2–3	=	MIDI port number (0–3)
R1	= 1	to enable touch sensitivity of interpreter.
	2	to disable touch sensitivity.
		Any other values of R1 ignored at present.

On exit

R0 undefined

Example

In BASIC:

```
SYS "MIDI_SoundEnable",1
```

does the same as:

```
*MidiSound in
```

MIDI_SetMode (&404C1)

Set the MIDI channel mode of the internal sound system controller. Use 0 to read current values.

On entry

R0		=	mode number, 1–4, if 0 then unchanged.
R1	byte 0	=	basic channel number N, 1–16, if 0 then unchanged.
	byte 1	=	number of channels in mode 4, 1–8 (M), if 0 then unchanged.

On exit

R0	=	new (or current) mode number (1–4).
R1	=	new (or last) settings of N (1–16) and M (1–8).

The possible modes are defined under Channel Modes.

MIDI_SetTxChannel (&404C2)

Set the MIDI channel and port number for subsequent transmitted commands to be sent on. This applies to all SWI commands prefixed by MIDI_Tx except MIDI_TxByte and MIDI_TxCommand.

On entry

R0 = new channel number (1–64), if 0 then unchanged.

1–16 MIDI channels 1–16 of receivers connected to MIDI port number 0.

17–32 channels 1–16 connected to port 1.

33–48 channels 1–16 connected to port 2.

49–64 channels 1–16 connected to port 3.

On exit

R0 = new (or current) channel number.

If the selected MIDI port is not installed, then it is undefined which port this and other SWIs will use instead. Use MIDI_Init to find the maximum port number installed, and never exceed it.

MIDI_SetTxActiveSensing (&404C3)

This puts the transmitter into Active Sensing mode, which causes dummy bytes to be transmitted in the absence of any other MIDI activity for longer than 280ms. The receiver should automatically switch to Active Sensing mode and expect this activity, or switch off all voices if it stops. This prevents voices becoming 'stuck on' if the MIDI cable becomes disconnected. Some MIDI receivers do not support this.

On entry

R0 bit 0 = 0 to stop automatic regular transmission of Active Sensing Messages in requested Port.

= 1 to start automatic regular transmission of Active Sensing Messages in requested Port.

bits 1–2 = Midi Port Number.

On exit

(This differs from the Archimedes MIDI Interpreter Version 2).

R0 bits 0–3 corresponding to Midi Ports 0–3; bits set if Transmit Active Sensing is enabled for this Midi Port.

bits 4–7 corresponding to Midi Ports 0–3; bits set if this Port is receiving Active Sensing.

MIDI_InqSongPositionPointer (&404C4)

Return the value of the internal Song Position Pointer, which is the value of the MIDI beat counter divided by six. See the chapter entitled *Timing*.

On entry

Unimportant.

On exit

R0	=	Song Position Pointer.
R1	=	bits set according to current state of MIDI:
bit 0		set if in External Timing Mode (Start message has been received).
bit 1		set if in Internal Timing Mode (Start message has been transmitted. Timing Clock transmission is automatic).
bit 2		set if in Fast Clock Mode.
bit 3		set if new (version 3) facilities enabled (`MIDI_FastClock` has been called). This flag is only reset on MIDI_Init with R0=0, `*Rminit midi` or Ctrl-Break.
bit 4		set if in special mode to store System Real Time Messages in receive buffer.
bit 5		set if in special mode to cause System Real Time Messages not to be treated in a special way.

Bits 0 and 1 are determined by the current timing mode. See the chapter *Timing*.

Bits 2 and 3 are set by calling MIDI_FastClock with relevant parameters.

Bits 4 and 5 are set by calling MIDI_Init with bits 30 and 31 of R0 set.

MIDI_InqBufferSize (&404C5)

Return the number of empty bytes in the transmit or receive buffer. These buffers can fill (rx buffer) or empty (tx buffer) at a maximum rate of 320 microseconds per byte. Default buffer sizes are:

- Transmit buffer size – 512 bytes
- Receive buffer size – 1024 bytes (programmable with MIDI_SetBufferSize).

On entry

R0	bit 0	= 0	to read rx buffer size.
		1	to read tx buffer size.
	bits 1–2	=	MIDI port number 0–3.

On exit

R0	=	number of bytes free in selected buffer.

MIDI_InqError (&404C6)

Return the value of the MIDI error bytes.

On entry

Unimportant.

On exit

R0 = up to four error bytes, corresponding to one byte per installed MIDI port.

Possible values of the error byte (shown as the ASCII character with the decimal value following in brackets) are:

'A' (65) Active Sensing failure error (MIDI connection removed, or Active Sensing transmission was stopped by the transmitter).

'B' (66) Receive data FIFO buffer was (and still may be) full and data has been lost. The application program should take the data more quickly.

'O' (79) UART overrun error. This means that the received data arrived, but was not read from the UART receive register before it was overwritten by the next data byte. This might occasionally happen when there is a lot of other processing occurring with the processor interrupt flag clear (high-numbered screen modes with simultaneous sound and intensive processor activity might also occasionally cause this error).

'F' (70) Framing error. This flag is generated by the UART when serial data arrives which does not fit the expected protocol, ie not sensible MIDI data.

'V' (86) Received MIDI data has caused the interpreter to attempt to use more than eight voices of the internal sound system, which are allocated on a first-come first-served basis.

'T' (84)	Transmit data FIFO buffer has overflowed, and data to be transmitted has been lost. The application program should transmit data more slowly, or use MIDI_InqTxBufferSize.
'L' (76)	Note too low (or too high) for the internal sound system received by the interpreter and ignored. The lowest note that the sound system can make is the C four octaves below middle C, or MIDI value 12. The highest note is MIDI value 96 or three octaves above middle C.
0 (zero)	No error.

NOTES:

- Only the latest error is shown. Previous errors are overwritten.
- Overrun and framing errors are also returned as standard SWI errors by MIDI_RxCommand and MIDI_RxByte at the time that they read the corrupted byte.
- The error is cleared when read.

MIDI_IgnoreTiming (&404DF)

This operates as a switch, instructing the system either to ignore any further received Timing messages: Start, Continue, Stop and Timing Clock, or to revert to normal reception of them.

On entry

R0 = 0 receive messages normally (default).

= 1 ignore received timing messages.

On exit

No change.

NOTE: There is a subtle distinction between the mode set by this SWI, where received Timing Messages are completely ignored, and the mode set by calling MIDI_Init with bit 31 set, which only disables special actions on Timing Message reception. The messages may still be stored in the receive buffer in the latter case, if MIDI_Init is called with bit 30 set, but not if Ignore Timing mode is set.

MIDI_SynchSoundScheduler (&404E0)

On entry

R0 = 0 set normal mode where sound scheduler is synchronised to the Sound Interrupt (SIRQ).

= 1 set special mode where the sound scheduler is synchronised to incoming MIDI Timing Clock Messages (prefixed by Start, ended by Stop). Scheduler time is incremented by one tick for each Timing Clock message received.

On exit

R0 = previous value of SynchSoundScheduler flag.

MIDI_FastClock (&404E1)

On entry

R0	= <0	read current value of fast clock.
	= 0	stop fast clock; revert to normal timing.
	= >0	= t = Timing Clock Transmission rate.
		Reset and start fast clock. Incoming data will be time stamped with the time in milliseconds shown on this clock. When started with MIDI_TxStart or *midistart, Timing Clock messages will be automatically transmitted at a rate of one every t milliseconds. The transmission will be stopped with SWI MIDI_TxStop or *midistop.
R1	=	time to reset clock to if R0 >= 0.

On exit

R1	=	previous value of fast clock

The fast clock increments every millisecond.

This SWI should be called once by new applications that want to use the MIDI scheduler. This ensures that software written for the old MIDI firmware will still work as before (it was previously insensitive to the value of R1 on entry to MIDI_TxCommand).

On calling MIDI_FastClock with R0 > 0, a new timing mode is set (Fast Clock Mode). In this Fast Clock Mode the value in R1 when calling SWI MIDI_TxCommand will determine the schedule time in milliseconds, as registered by the Fast Clock. If the value in R1 is zero, then the message will be sent immediately.

IMPORTANT NOTE: The Midi scheduler queue is cleared and reset on calling MIDI_FastClock with 1 <= r0 <= &FFFF, therefore this should not be called if there are Note Off messages pending in the Scheduler, otherwise notes could be left hanging on indefinitely in the receiver.

MIDI_Init (&404E2)

On entry

R0 = 0 do an Internal System Reset (reset to power-on state).

= >0 bit 0 set to clear current transmitted running status (ensure status is included with next transmitted message).

bit 1 set to clear receive buffers and reset midi interpreter.

bit 2 set to clear transmit buffers and reset midi interpreter.

bit 3 set to clear MIDI Scheduler.

bit 4 set to clear current error.

(special mode bits, only cleared by a call with R0=0).

bit 30 set to enable special mode so that received System Real Time messages will be stored in the receive buffer, so that they can be read with SWI MIDI_RxCommand and SWI MIDI_RxByte.

bit 31 set to prevent special actions on reception of System Real Time messages.

On exit

R0 = number of recognised Midi Ports installed.

Subtract one from this number for the maximum port number, which should not be exceeded.

Certain MIDI recording and replaying applications may need to set bits 30 and 31 so that they can precisely reproduce the recorded data with the Real Time messages.

MIDI_SetBufferSize (&404E3)

Clears the buffer and then claims the requested buffer size from the RMA. Returns `No room in RMA` error (&102) if unable to claim the new buffers, and leaves the previous buffers intact.

NOTE: This should only be used when the buffers are empty, otherwise data will be lost.

On entry

R0	=	0 for set receive buffer size (nothing else currently supported).
R1	=	new buffer size in bytes.
	= 0	interrogate current value.

On exit

R0	=	buffer size in bytes.
R1	=	total space in bytes claimed from RMA for new buffers.
	=	5 x size x number of MIDI Ports installed for receive buffer (for each byte received, four bytes of time stamp is also stored)

MIDI_Interface (&404E4)

(For advanced use only.) Get addresses for more efficient access to critical SWIs.

On entry

R0 = workspace pointer, moved to R12 when calling a SWI through this interface.

R1 = SWI code pointer.

When calling SWIs using this interface, the CPU should be in supervisor mode.

The MIDI SWIs do not support re-entrancy, so they should not generally be called from an interrupt routine. R11 should contain the SWI offset from chunk base (MIDI_SoundEnable = 0). R12 should contain the workspace pointer. The addresses become invalid if the MIDI module is re-initialised, or finalised, so watch for MIDI service calls to warn of this, and re-call this SWI.

MIDI_RxByte (&404C7)

Return the next received MIDI byte, excluding Real Time messages which are processed internally (see section on special messages). In general, MIDI_RxCommand should be used in preference to this command, for reduced SWI time overhead, although this SWI should be used for reading the 'raw' data.

On entry

R0	=	port number (0–3) to receive message from.
	= –1	to look at each port in order of increasing port number, until one is found in which new data has been received, and return that data.

On exit

R0 byte 0	=	next received MIDI byte.
	= 0	if receive buffer empty, or incomplete message received. If entered with R0= –1, distinguish between these cases by checking the port number returned in bits 28–31.
bits 24–27=1		if byte received.
bits 28–31		MIDI port number where this byte was received.
R1	=	received time (see section on timing).
	= 0	if receive buffer empty or clock disabled.

In the case of a low-level error registered by the UART, this SWI returns an error number:

- &20402 if there was a Framing Error when this byte was received
- &20403 if there was an Overrun Error when this byte was received.

The error number &20404 will be returned if the receive buffer overflowed.

The corrupted byte will be returned on the next SWI Rx.

MIDI_RxCommand (&404C8)

Return the next complete MIDI command as a set of bytes (normally excluding System Real Time). A status byte should always be returned, even when the incoming data is on running status, except when the receive buffer is empty.

NOTE: System Exclusive messages will be received as one status byte (&F0) and one data byte, and successive calls to MIDI_RxCommand will treat the system exclusive as Running Status, with one data byte, until EOX (end of exclusive: &F7) or any other status byte, except System Real Time, is encountered (&80–&F7).

On entry

R0	=	port number (0–3) message will be received from.
	= –1	look at each port in order of increasing port number, until one is found in which new data has been received, and return that data, if Midi Message entirely received, or 0 if not or if no port has received any data.

On exit

R0	byte 0	=	status.
	byte 1	=	data byte 1, or 0 if none expected.
	byte 2	=	data byte 2, or 0 if none expected.
	bits 24–27		number of bytes in this message = 0–3.
	bits 28–31		MIDI port number of port that this message was received by.
		= 0	if receive buffer empty, or incomplete message received.
R1		=	received time of last byte in message.
		= 0	if receive buffer empty or clock disabled.

In the case of a low-level error registered by the UART, this SWI returns the following error numbers:

- &20402 if there was a Framing Error when a byte in this message was received.
- &20403 if there was an Overrun Error when a byte in this message was received.

The error number &20404 will be returned if the receive buffer overflowed.

The corrupted byte will be returned on the next MIDI_RxByte or MIDI_RxCommand.

Byte 3 returned in R0 is the total number of bytes comprising the command (excluding the time stamp), and thus is the number of data bytes +1. The MIDI specification details the number of data bytes with each command, which is thus different by one.

MIDI_TxByte (&404C9)

Transmit a byte from the MIDI OUT. This will be transmitted, regardless of Running Status.

On entry

R0 byte 0 — Byte to transmit.

bits 28–31 — Port number to transmit from (0–3).

On exit

Unchanged.

Returns error number &20401 if this fails because the transmit buffer is full.

Data transmission commands with automatic running status optimisation

Automatic Running Status optimisation applies to all the commands in this section; that is, the status byte may not be transmitted if the last command had the same status. However, a limit applies to long chains of commands under Running Status, and the status byte is periodically retransmitted.

MIDI_TxCommand (&404CA)

Transmit or schedule on the MIDI schedule queue a complete MIDI command. The status byte may not be transmitted if Running Status applies. The command will be ignored and not transmitted if byte 0 is not a status byte (bit 7 set), or if the data bytes have bit 7 set.

NOTE: The format is the same as MIDI_RxCommand, so that received commands can be simply retransmitted without decoding or encoding. It can be used to transmit a MIDI command immediately, or to schedule one to be transmitted at some future time (if in Fast Clock mode, and R1>0).

On entry

R0	byte 0	=	status.
	byte 1	=	data byte 1, if required by specified status.
	byte 2	=	data byte 2, if required by specified status.
	bits 24–28		(optional) number of bytes in command; only needed for status undefined in MIDI Specification 1.0.
	bits 28–31		port number to transmit from (0–3.)
R1		=	schedule time (0 for immediate) in:
			Fast clock time if in Unset or Internal timing modes.
			Timing Clock received count time if in External timing mode.

On exit

If R1 was non-zero on entry:

R0 = number of scheduler slots free in queue.

= –1 if it failed because the scheduler was full.

= –2 if it failed because the schedule time requested was earlier than the time of the previous event in the scheduler queue.

Returns error number &20401 if this fails because the transmit buffer is full.

All commands will be transmitted in the current MIDI transmission channel defined by the last MIDI_SetTxChannel.

NOTES:

- The size of the scheduler queue is 1024 commands.
- All calls to RxCommand with non-zero R1, from clearing the scheduler should be with non-decreasing schedule time. For efficiency, only the schedule time of the next item on the queue is inspected at each time increment, and items are removed from the queue in the order they were put in.
- For backwards compatibility, the value of R1 on entry to this routine is ignored if SWI MIDI_FastClock has not been called since the module was initialised. This state can be interogated with MIDI_InqSongPositionPointer. R1, bit 3 is zero if in 'backwards compatible' mode.

MIDI_TxNoteOff (&404CB)

Transmit a MIDI Note Off command.

On entry

R0	=	note number, 0-127 (60 = middle C, 1 unit = 1 semitone).
R1	=	key off velocity, 0-127.

On exit

No change.

MIDI_TxNoteOn (&404CC)

Transmit a MIDI Note On command.

On entry

R0	=	note number, 0-127 (60 = middle C, 1 unit = 1 semitone).
R1	=	key on velocity, 0-127.

On exit

No change.

MIDI_TxPolyKeyPressure (&404CD)

Transmit a MIDI Poly Key Pressure (after touch) command. This will apply to any one voice and may affect volume, modulation or pitch depending on the setting of the receiver.

On entry

R0 = note number, 0-127 (60 = middle C, 1 unit = 1 semitone).

R1 = key pressure value, 0-127.

On exit

No change.

MIDI_TxControlChange (&404CE)

Transmit a MIDI Control Change command.

On entry

R0 = control number 0-121, 122-127 reserved for channel mode messages, which can be transmitted using this command, or by using the Channel Mode SWIs; see the chapter *MIDI interface data specification*.

On exit

No change.

MIDI_TxLocalControl (&404CF)

Transmit a MIDI Local Control command with control number of 122, and R0 having one of the values defined below.

On entry

R0 = 0 local control off.

R0 = 127 local control on.

On exit

No change.

MIDI_TxAllNotesOff (&404D0)

Transmit a MIDI All Notes Off command.

MIDI_TxOmniModeOff (&404D1)

Transmit a MIDI Omni Mode Off command.

MIDI_TxOmniModeOn (&404D2)

Transmit a MIDI Omni Mode On command.

MIDI_TxMonoModeOn (&404D3)

Transmit a MIDI Mono Mode On command.

On entry

R0 = M where M is the current number of channels 1-16.

On exit

No change.

MIDI_TxPolyModeOn (&404D4)

Transmit a MIDI Poly Mode On command.

MIDI_TxProgramChange (&404D5)

Transmit a MIDI Program Change command. The program referred to is the 'voice', 'tone' or 'patch' number in the receiver.

On entry

R0 = program number, 0-127.

On exit

No change.

MIDI_TxChannelPressure (&404D6)

Transmit a MIDI Channel Pressure command. This will affect all the notes on that channel, and may alter volume, modulation or pitch depending on the receiver setting.

On entry

R0 = pressure value, 0-127.

On exit

No change.

MIDI_TxPitchWheel (&404D7)

Transmit a MIDI Pitch Wheel command.

On entry

R0 = pitch wheel change, 0-16383 (&3FFF). 8192 (&2000) is centre position value (no pitch change).

On exit

No change.

MIDI_TxSongPositionPointer (&404D8)

Transmit a MIDI Song Position Pointer command. This automatically updates the internal copy which could affect the time-stamping of received data.

On entry

R0 = Song Position Pointer, 0-16383 (&3FFF).

On exit

No change.

MIDI_TxSongSelect (&404D9)

Transmit a MIDI Song Select command.

On entry

R0 = song number, 0-127.

On exit

No change.

MIDI_TxTuneRequest (&404DA)

Transmit a MIDI Tune Request command.

MIDI_TxStart (&404DB)

Transmit a MIDI Start command, reset the MIDI beat counter to zero (and the internal Song Position Pointer), enable automatic transmission of Timing Clock messages every 16 internal microbeats (or else timed by fast clock; see MIDI_FastClock), and disable reception of Start, Continue, Stop and Timing Clock messages. This affects all installed MIDI ports.

MIDI_TxContinue (&404DC)

The same as TxStart, but without resetting the beat counter.

MIDI_TxStop (&404DD)

Transmit a MIDI Stop command, and stop transmission of Timing Clock messages, thus allowing reception of Start, Continue, Stop and Timing Clock messages.

MIDI_TxSystemReset (&404DE)

Transmit a MIDI System Reset command.

Service_MIDI (&58)

Service call.

Sub reason codes

`Service_MIDIAlive = 0`	module about to be initialised.
`Service_MIDIDying = 1`	module about to be removed.

Enters module service with R1 = Service_MIDI, and R0 = sub reason code.

Event_MIDI (&11)

Sub reason codes

`MIDI_DataReceivedEvent = 0`	receive buffer was empty and has received some data.
`MIDI_ErrorEvent = 0`	an error has occurred in the background; use MIDI_InqError to find out what it is.
`MIDI_ScheduleEmptyingEvent = 2`	the MIDI scheduler will empty within the next 10ms. This event does not occur in cases where there are fewer than four free slots in the scheduler when the event would normally be triggered, which will only occur if more than 1020 commands are scheduled to happen, all within 10ms.

Enters an event handler routine with R0 = Event_MIDI and R1 = sub reason code.

Summary of SWI commands

SWI	Number
MIDI_SoundEnable	&404C0
MIDI_SetMode	&404C1
MIDI_SetTxChannel	&404C2
MIDI_SetTxActiveSensing	&404C3
MIDI_InqSongPositionPointer	&404C4
MIDI_InqBufferSize	&404C5
MIDI_InqError	&404C6
MIDI_RxByte	&404C7
MIDI_RxCommand	&404C8
MIDI_TxByte	&404C9
MIDI_TxCommand	&404CA
MIDI_TxNoteOff	&404CB
MIDI_TxNoteOn	&404CC
MIDI_TxPolyKeyPressure	&404CD
MIDI_TxControlChange	&404CE
MIDI_TxLocalControl	&404CF
MIDI_TxAllNotesOff	&404D0
MIDI_TxOmniModeOff	&404D1
MIDI_TxOmniModeOn	&404D2
MIDI_TxMonoModeOn	&404D3
MIDI_TxPolyModeOn	&404D4
MIDI_TxProgramChange	&404D5
MIDI_TxChannelPressure	&404D6

MIDI_TxPitchWheel	&404D7
MIDI_TxSongPositionPointer	&404D8
MIDI_TxSongSelect	&404D9
MIDI_TxTuneRequest	&404DA
MIDI_TxStart	&404DB
MIDI_TxContinue	&404DC
MIDI_TxStop	&404DD
MIDI_TxSystemReset	&404DE
MIDI_IgnoreTiming	&404DF
MIDI_SynchSoundScheduler	&404E0
MIDI_FastClock	&404E1
MIDI_Init	&404E2
MIDI_SetBufferSize	&404E3
MIDI_Interface	&404E4

Programming with software interrupts

In simple BASIC programs, the SWIs may be called as in the following example:

```
SYS "MIDI_TxNoteOn",60,64
```

The two parameters are the values of R0 and R1. Case is important – upper and lower case must be used as above. The most efficient way to use these SWIs in BASIC is to define integer variables at the start of the program.

To find the SWI number from the string, OS_SWINumberFromString:

```
SYS "OS_SWINumberFromString",0,"MIDI_TxNoteOff" TO
TxNoteOff%

SYS "OS_SWINumberFromString",0,"MIDI_TxNoteOn" TO
TxNoteOn%
```

```
SYS "OS_SWINumberFromString",0,"MIDI_InqError" TO
InqError%
```

Then to use them:

```
v%=64

a%=60

SYS NoteOn%,   a%, v%

key% = INKEY (80)

SYS NoteOff%,  a%, v%

SYS InqError%,0 TO Error%

IF Error% <>0 THEN PRINT CHR$(Error%)
```

(the line beginning `key%` provides a delay of 80 centiseconds).

Timing

Timing

The timing mode can be:

- Unset
- Internal
- External.

Internal timing mode is set if timing messages are being transmitted.

External timing mode is set if timing messages are currently being received and not transmitted (or ignored).

Three possible types of time-stamp can be returned by RxCommand and RxByte depending on which of four internal timing modes is current. These four modes are:

- MIDI Timing Clock transmission counting. The time-stamp is the value of the MIDI beat counter, which is equal to the number of Timing Clock messages transmitted since the last Start transmitted.
- MIDI Timing Clock reception counting. The time-stamp is the value of the MIDI beat counter equal to the number of Timing Clock messages received since the last received Start command.
- A3000 computer sound system beat counter (see the chapter on sound in the *A3000 User Guide*). The time-stamp is the value of the sound system bar/beat counter. The bar counter is incremented every time the beat counter resets to zero, and the time-stamp is a combination of the sound system beat value and an incrementing bar value.

  The beat count is off by default, so BEATS must be set to a positive value to set the beat count in operation, and so give non-zero values for internal BAR/BEAT time-stamping.
- Fast Clock Timing mode. The Fast Clock is reset and started by SWI MIDI_FastClock, and incremented every millisecond. The timestamp is set equal to this count.

NOTE: The types of time-stamp are related to the different time periods, or beats, used. For the purposes of this guide, an A3000 computer sound system beat, as defined in the *A3000 User Guide*, is referred to as a microbeat, in order to distinguish it from a MIDI beat.

One MIDI beat is equal to 16 microbeats, when microbeats govern the timing of MIDI beats.

MIDI Timing Clock transmission counting

This mode is set by the call MIDI_TxStart, if the fast clock is not running, and any received Start, Continue, Stop or Timing Clock messages are ignored. Incoming data is time-stamped with the current value of the MIDI beat counter, the value being returned in R1 on calling MIDI_RxCommand, or MIDI_RxByte.

The call MIDI_TxStart enables automatic transmission of Timing Clock messages. They can then be stopped with MIDI_TxStop, and restarted with MIDI_TxContinue. The frequency can be controlled by programming the sound system tempo, using Sound_QTempo (or TEMPO in BASIC). A MIDI Timing Clock message will be transmitted every 16 sound system microbeats. The song position pointer is equal to the value of the MIDI beat counter divided by six.

For example, if the sound system is programmed to the default tempo of &1000 (=100 per second), then, when enabled by MIDI_TxStart, Timing Clock messages will be transmitted at a rate of 6.25 per second, and the song position pointer will increment approximately once per second. For the maximum tempo of &FFFF the pointer will increment at a rate of 16.66 per second and Timing Clock messages will be transmitted at a rate of 100 per second.

NOTE: The sound system beat counter must be started to enable this mode to work (for example, BEATS 100 in BASIC).

MIDI Timing Clock reception counting

When Timing Clock messages are being transmitted, they take priority. If not, any Timing Clock messages received (preceded by Start) will cause the MIDI beat counter to increment. This can be prevented by disabling reception of timing messages, using MIDI_IgnoreTiming with R0=1.

The MIDI beat counter is updated automatically on reception of a Song Position Pointer message only if the MIDI beat counter is currently being controlled by received Real Time messages. The MIDI beat counter is set to the received value multiplied by six. The internal MIDI beat counter is always updated by the call MIDI_TxSongPositionPointer.

Sound system bar/microbeat counting

If Fast Clock Timing mode is not enabled, and if Timing Clock messages are neither being received nor transmitted, which may be because they were never started, or because they were stopped with a MIDI Stop command, then the received bytes are given a sound system bar/microbeat time stamp, with the value of microbeat in the bottom 16 bits, and an incrementing bar count in the top 16 bits. The bar counter is reset on transmission or reception of a MIDI Stop, or on changing the value of BEATS (microbeats per bar), or upon incrementing beyond &FFFF.

Queuing of MIDI data to be transmitted on a future microbeat can be done using the general-purpose sound system call:

Sound_QSchedule

On entry: R0 = the schedule period.
R1 =&F000000 + SWI number to schedule.
R2 =SWI parameter for R0.
R3 = SWI parameter for R1.

On exit: R0 =0 if SWI successfully added to queue.
R0 <0 if queue is full and command failed.

NOTE: The queuing time is the value of the sound system microbeat counter and NOT the MIDI beat counter.

For details of the microbeat timing see the *A3000 User Guide*.

The Sound Scheduler may be synchronised to an external source of MIDI Timing Clock messages by using SWI MIDI_SynchSoundScheduler with R0 = 1.

NOTE: For new applications it is preferable to use the MIDI Scheduler by calling SWI MIDI_TxCommand with R1 = Schedule time (see Fast Clock Timing mode below).

Fast Clock Timing mode

This mode is enabled by calling SWI MIDI_FastClock. Incoming messages will be stamped with the current Fast Clock time, unless timing messages are being received, in which case MIDI Timing Clock Reception Counting (see above) will apply. Scheduling to this time is done using the special MIDI Scheduler by calling the SWI MIDI_TxCommand with the schedule time in R1.

Reception of special messages

Most MIDI messages when received have no further effect than to be stored in the receive buffer to be read by the SWIs MIDI_RxCommand and, MIDI_RxByte, or by the MIDI interpreter. Certain messages, however, have other side-effects, and some are not stored in the receive buffer at all, making them invisible to the application, except by their side-effects. This behaviour can be reprogrammed if required using MIDI_Init with special bits set.

Song Position Pointer as a side-effect updates the internal MIDI beat counter to the received value multiplied by six.

System Real Time messages, except for System Reset, are invisible to the application except in their side-effects.

Real Time messages

These Real Time messages are:

Start	If Timing Clock messages are not being transmitted, ie the call MIDI_TxStart has not been made since the last Stop, then reset the internal MIDI beat counter, and the Song Position Pointer) and enable received Timing Clock messages to increment it.
Timing Clock	If external timing is enabled (see Start) then increment the MIDI beat counter. The Song Position Pointer is incremented on every sixth Timing Clock message received.
Stop	Disable external clocking. Received Timing Clock will have no further effect.
Continue	The same as Start, without resetting the MIDI beat counter.
Active Sensing	This will set the receiver into Active Sensing mode, so that if no message is received for more than 300ms for any reason (for example if the MIDI cable is pulled out), then all MIDI-triggered notes are turned off. The transmitter should send regular Active Sensing status messages, when enabled, in the absence of any other activity. An application program can be warned of an Active Sensing timeout by polling the error flag, or by enabling the MIDI error event.

Exceptions

It is the task of the application program to deal with exceptions and errors.

Exceptions

It is the task of the application program to deal with exceptions and errors. For example, when Note On messages have been sent via the MIDI, and then an error occurs, or escape is pressed, the program must automatically send a matching set of Note Off messages, to avoid notes becoming stuck on. For this reason, if you are using the Sound Scheduler to schedule MIDI data, use Sound_QInit only with great care to avoid losing Note Off messages and hearing notes in the receiver.

Removing sound modules

If a sound module is removed for any reason, the MIDI module may get into an undefined state. Replace the sound module and type `*RMInit MIDI` if this occurs.

MIDI interface data specification

Summary specification

MIDI equipped instruments contain a receiver and a transmitter, the receiver being optoisolated. It interprets and acts on MIDI commands. The transmitter sends messages in MIDI format via a line driver.

Data rate

31.25 (+/– 1%) Kbaud, asynchronous. There is a start bit, eight data bits and a stop bit, so each serial byte consists of 10 bits lasting 320 microseconds.

Summary of status bytes

	Status D7–D0	No. of Data bytes	Description
Channel voice messages	&8n	2	Note Off
	&9n	2	Note On (velocity=0, note off)
	&An	2	Polyphonic Key Pressure (after touch)
	&Bn	2	Control Change if data byte 1 in the range 0–121
	&Cn	1	Program Change
	&Dn	1	Channel Pressure (after touch)
	&En	2	Pitch Wheel Change
Channel mode messages	&Bn	2	Channel mode message if data byte 1 in the range 122–127
System messages	&F0	any	System Exclusive
	&Fs	0 to 2	System Common
	&Ft	0	System Real Time

Where n = N–1; N is channel number.
s = 1 to 7
t = 8 to 15

User Port

General

The User Port is compatible with the User port on the I/O expansion card for the Archimedes computer, and largely compatible with the User Port fitted to BBC Model B and Master 128 computers, and enables you to connect your A3000 computer to the wide range of peripheral equipment already available for these computers.

It consists of 8 data lines and two control lines from half of a 65C22 Versatile Interface Adapter chip (VIA). The VIA contains 16 internal registers, and these are mapped into memory. On the BBC Microcomputer and on the A3000 computer, legal access to these registers is made by using the two OSBYTE calls which read and write to SHEILA, numbers 150 and 151.

The signals available on the connector are the 8 data lines PB0 to PB7 on pins 6, 8, 10, 12, 14, 16, 18 and 20 respectively, and the two interrupt/handshake/shift register control lines CB1 and CB2 on pins 2 and 4 respectively.

When used for data transfer using handshaking, the CB2 signal is a 'data ready' output to the peripheral, and the CB1 signal is a 'data taken' input from the peripheral.

When used in interrupt mode, CB1 and CB2 cause the IRQ line of the VIA to go low. However, interrupts from the VIA are not normally supported. See below.

Serial data can be shifted into or out of the CB2 pin under control of either an internal timer or from an external clock applied toCB1.

The A3000 implementation

The pin-out of the User Port 20-pin IDC connector is shown overleaf:

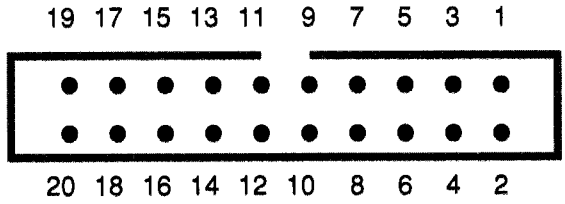


figure 4: User Port pin-outs, viewed from rear

1 +5v	2 CB1	11 0v	12 PB3
3 +5v	4 CB2	13 0v	14 PB4
5 0v	6 PB0	15 0v	16 PB5
7 0v	8 PB1	17 0v	18 PB6
9 0v	10 PB2	19 0v	20 PB7

The User Port is implemented using half of a 65C22 VIA chip. As on the BBC Microcomputer, the VIA registers are memory mapped, and control is exercised in the same way through OSBYTE calls 150 and 151, which read from and write to the I/O page SHEILA. It cannot be assumed that SHEILA is mapped into memory at a specific location, so direct access to the User Port through writing to or reading from specific addresses does not work.

Incompatibilities with the BBC User Port

The VIA chip on the expansion card is running at 2MHz instead of the BBC Microcomputer's 1MHz device. This means that the internal timers of the VIA are running twice as quickly as expected. If the shift registers are being used under control of the internal timer, then these too run twice as fast.

Power which may be taken from the User Port must not exceed 500mA.

The interrupt signal from the VIA is supported, but a suitable interrupt handler for the A3000 computer must be written.

Using the interface

Legal commands

The interface must be used through the legal BASIC and RISC OS commands. Any software which tries to access specific memory locations in the earlier BBC Microcomputer I/O space will not work.

OSBYTE calls 150 and 151, however, use the 6502 registers on the BBC Microcomputer and so are implemented slightly differently on the A3000 computer.

In general, parameters passed in A on the BBC Microcomputer are passed in the least significant byte of R0 on the A3000 computer. Those passed in X are now passed in the LSB of R1, and those passed in Y are now passed in the LSB of R2.

*FX commands still work as on the BBC Microcomputer, the parameters being passed in the correct registers automatically.

The legal commands are as follows:

- OSBYTE 150 Read a byte from SHEILA
- OSBYTE 151 Write a byte to SHEILA.

The 16 VIA registers which are memory mapped to the SHEILA I/O space have offsets &60 to &6F hex (96 to 111 decimal).

On entry: R0 contains the OSBYTE number.

R1 contains the offset in SHEILA.

R2 contains the byte to be written (for the write command).

On exit: R2 contains the byte which was read (for the read command).

An example

The User Port is controlled via the 16 registers of the VIA chip which are mapped into the I/O space SHEILA at offsets &60 to &6F. For example, to write &FF to DDRB (data direction register B).

```
10 osbyte%=6 :REM SYS 6 is equivalent to osbyte
20 writebyte%=151 :REM osbyte number for write byte to
   SHEILA
30 offset%=&62 :REM offset in SHEILA of DDRB
40 byte%=&FF :REM byte to put in DDRB
50 SYS osbyte%,writebyte%,offset%,byte%
```

The same example in assembly language is shown below:

```
10 REM write &FF to User Port DDRB
20 osbyte%=6 :REM SWI 6 is equivalent to OSBYTE
30 writebyte%=151 :REM osbyte number for Write byte to
   SHEILA
40 offset%=&62 :REM offset in SHEILA of DDRB
50 byte%=&FF :REM byte to put in DDRB
60 DIM code% 100
70 P%=code%
80 [
90 STMFD R13!,{R0-R12,R14} \ save registers on stack
100 MOV R0,#writebyte%   \ put osbyte number in R0
110 MOV R1,#offset%     \ put offset in R1
120 MOV R2,#byte%       \ put byte to be written in R2
130 SWI osbyte%          \ execute osbyte call
140 LDMFD R13!,{R0-R12,PC} \ pull registers from stack
    \ and return to BASIC
150 ]
160 CALL code%
```

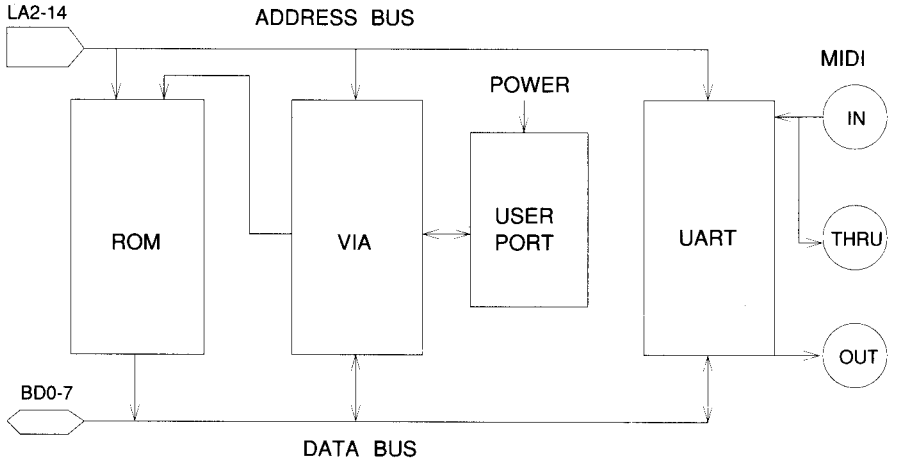
Technical specification

	Min	Typ	Max	Units
Operating voltage	4.5	5.0	5.5	V
Supply current to expansion card		50	100	mA
	+ current supplied from User Port			
Output drive capability			1	TTL i/p
Input load			1	TTL i/p
Output current (+5V)			500	mA

User Port address allocation

There is a SWI instruction which returns the absolute address location of the User Port / MIDI upgrade in the memory map. The SWI can be called either with its name (I/O_Podule_Hardware) or its number (&40500). On exit R1 contains the base address of the upgrade hardware. All other registers are preserved. The user port VIA is &2000 above this base address, and the VIA registers are four bytes apart.

Appendix A: Block diagram



Bibliography

MIDI

The *MIDI Specification Document* (no. MIDI-1.0, August 5th, 1983), is available from:

International MIDI Association
11857 Hartsook Street
North Hollywood
CA91607
USA.

This specification is included in:

MIDI for Musicians by Craig Anderton, AMSCO Publications (ISBN 0.8256.1050.8).

User Port

65C22 VIA Data Sheet (Rockwell).

Reader's Comment Form

A3000 User Port / MIDI User Guide

We would greatly appreciate your comments about this Guide, which will be taken into account for the next issue:

Did you find the information you wanted?

Do you like the way the information is presented?

General comments:

If there is not enough room for your comments, please continue overleaf

How would you classify your experience with computers?

☐ **First-time user** ☐ **Used computers before** ☐ **Experienced user** ☐ **Programmer**

Cut out (or photocopy) and post to:

Dept RC, Technical Publications
Acorn Computers Limited
645 Newmarket Road
Cambridge CB5 8PB.

Your name and address:

This information will only be used to get in touch with you in case we wish to explore your comments further.